Microcontroladores Estándar en el Desarrollo de Sistemas de Lógica Difusa

Sr. Arístides Bouza[*] - Ing. Omar Alimenti[**]
Dto. Ing. Eléctrica / Instituto de Ingeniería y Ciencias en Computación
Universidad Nacional del Sur
e-mail:iealimen@criba.edu.ar

Resumen

Un Sistema de Lógica Difusa (SLD) permite manejar datos numéricos y conocimientos lingüísticos en forma simultánea mediante una matemática unificada.

Los grandes avances tecnológicos han dado lugar al desarrollo de diferentes alternativas para la implementación de un SLD tales como el empleo de microcontroladores estándar con soporte de software para Lógica Difusa, coprocesadores "fuzzy" dedicados, procesadores RISC con soporte "fuzzy" y hardware orientado (ASIC).

Todo esto, ha ocasionado que las aplicaciones de control empleando Lógica Difusa, hayan alcanzado un gran desarrollo en la actualidad.

En este trabajo se desarrollan los conceptos básicos para implementar un Sistema de Lógicas Difusas ("kernel difuso") que optimiza la relación complejidad computacional vs. funcionalidad de los procesos de inferencia. Este kernel utiliza una base de conocimientos que relaciona directamente la entrada y salida "crisp". Esto, permite que plataformas naturamente limitadas aborden la solución de problemas más complejos.

Por último se describe una aplicación práctica que demuestra las cualidades del método.

Palabras Clave: Inferencia, antecedente, consecuente, fuzzificación, zona transicional, kernel difuso, centroide.

* Alumno de Proyecto Final de Carrera Ing. Electrónica - U.N.S.
** Profesor Asociado Lab. Sistemas Digitales - U.N.S.

1. Introducción

Normalmente un sistema de lógica difusa se nos presenta, en primera instancia, como una forma sencilla de resolver un problema. De hecho lo asimilamos mucho a la "solución que uno tendría en mente". También es conocida la complejidad computacional que los SLD traen aparejados, según se torna abstracta su definición.

Es frecuente cometer el error de creer ilimitadas sus posibilidades, haciendo de lado aspectos tales como:

- Plataforma física de implementación
- Tiempo de cómputo de la solución
- Tiempo de respuesta requerido

Otro aspecto a tener en cuenta, es la precisión que, naturalmente, la plataforma seleccionada nos ofrece [2]. Ésta, será en general, bastante pobre, disponiendo normalmente de un ancho de palabra limitado.

Con esta breve descripción ya podemos ver que para lograr resultados realistas, no se pueden dejar librados al azar ni siquiera los temas que no hacen estrictamente al sistema de lógica difusa, ya que estos pueden afectar la performance del sistema.

En este trabajo se ha desarrollado un método que tiene en cuenta los factores anteriormente descriptos, para lograr balancear tiempo de cómputo con calidad de resultados. Este fue ensayado sobre un sistema práctico consistente en un par motor-generador para mantener constante su velocidad [6] ante variaciones de carga, obteniéndose resultados muy satisfactorios.

2. Sistemas de Lógica Difusa

Los tipos de implementaciones que aquí trataremos son muy ceñidos a la actividad ingenieril, en el sentido que emplean una cantidad restringida de recursos matemáticos intentando minimizar la cantidad de cálculos para obtener en un tiempo "razonable" una solución "adecuada" [1].

De acuerdo al párrafo anterior, comenzamos describiendo nuestro sistema como un mapeo no lineal de una o más cantidades numéricas (entradas) en un escalar (salida).

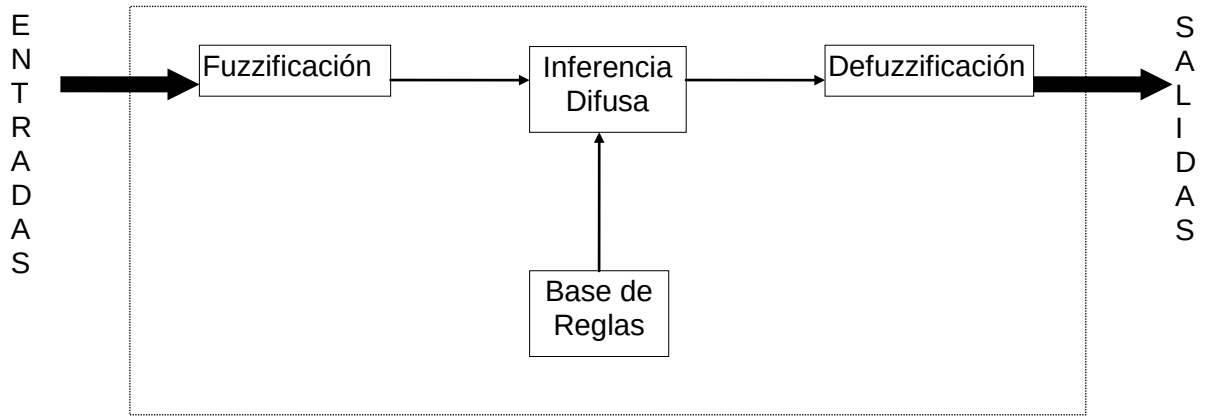


Figura 1

Esta figura, nos da una idea más clara respecto a la disposición interna del sistema. Es importante comprender, que nuestras entradas y salidas son números concretos y no conjuntos abstractos que resultan de la inferencia difusa.

Internamente, nuestro sistema tipifica la entrada de acuerdo a su similitud (de aquí en adelante "membresía") con uno o más de los conjuntos de entrada. Este proceso es realizado en la caja denominada "Fuzzificación" (ver Fig. 1) y su actividad es observar la

entrada buscando si determinada característica de la misma coincide con la de algún conjunto de entrada y en que medida. Por ejemplo un setter y un ovejero alemán no resultan iguales pero tienen características comunes como: ser perros, tener cuatro patas, etc., vemos entonces que hay algún grado de membresía entre la descripción de un setter y un ovejero alemán.

Las reglas son sencillas operaciones de inferencia lógica con las que se describe el comportamiento del sistema y que se asimilan a una estructura tipo SI - ENTONCES...

SI (se cumplen determinadas causas) **ENTONCES** (ocurre tal consecuencia).

Es justamente ésta una faceta casi única que los sistemas de lógica difusa tienen, cual es la de poder incorporar conocimientos de tipo cualitativo/subjetivo (además de los tradicionales). Dicha posibilidad habilita la incorporación del conocimiento del experto al sistema en su forma nativa (ie: en forma cualitativa).

La inferencia difusa es una adaptación de la tradicional para operar con variables y conjuntos de variables difusas.

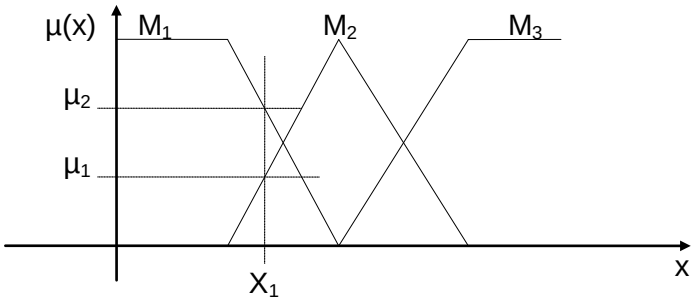
Finalmente, la defuzzificación es la operación que coordina los valores difusos, resultantes de la inferencia, para dar un valor numérico concreto a la salida como era nuestro interés.

3. Fuzzificación

La *fuzzificación* es el proceso por el cual se "traduce" una variable numérica (en general medida en el sistema físico que se pretende controlar), en una concepción difusa de dicho valor. Este proceso presupone:

- que los valores numéricos de la variable se encuentran en un dominio conocido o al menos acotado.
- que dicho dominio ha sido debidamente particionado en uno o más subconjuntos cuya unión abarca el dominio completo.
- que a cada subconjunto se le ha asignado un nombre relacionado con la propiedad que lo caracteriza y una función de pertenencia al mismo.

Este proceso toma el valor numérico de entrada y obtiene el grado de pertenencia de la variable a uno o varios de los subconjuntos definidos en su dominio.



x: Variable numérica de entrada
X_1:valor concreto a fuzzificar
μ_1,μ_2:pertenencias del valor X_1 a los conjuntos M_1,M_2

Figura 2

El fuzzificador de la figura 2 es llamado ***singleton*** (expresión inglesa de simple o simplón). No es el único de su clase, pero resulta sumamente útil en aplicaciones de ingeniería, ya que consiste en operaciones sencillamente computables. Otros métodos de fuzzificación (con mayores costos de cómputo) dan como resultado un conjunto difuso en vez de un valor de membresía. Los conjuntos de salida de dichos métodos pueden ser del estilo de las funciones de la figura 2 ó de otros tipos (Gaussianos,etc...) y son, en general, representativos de la incerteza acerca del valor de la variable numérica de entrada.

4. Inferencia o razonamiento

En primera instancia deberíamos responder las siguientes preguntas:

1ro: *"¿Qué es una proposición?"*

Una proposición es una aseveración que puede ser verdadera o falsa, total o parcialmente.

2do: *"¿Qué es una regla?"*

Una regla es un conjunto de proposiciones combinadas entre sí, de modo tal que si se cumple un grupo de proposiciones de partida llamadas antecedentes, debe ser entonces lógicamente otro grupo de proposiciones de llegada, llamadas consecuentes.
Finalmente sean:

- un conjunto de reglas dado
- un cierto grupo de antecedentes
- un cierto grupo de consecuentes

En este contexto, diremos que inferencia es el proceso por el cual evaluando la veracidad de los antecedentes y teniendo en cuenta las reglas, se deduce la veracidad de los consecuentes.

Todas las maneras que hay de efectuar estas operaciones, se derivan de las siguientes:

- La **conjunción lógica** ($p \wedge q$) en la que se establece la veracidad simultánea de dos proposiciones.
- La **disyunción lógica** ($p \vee q$) en la que se establece la veracidad de una de dos proposiciones.
- La **implicación lógica** ($p \Rightarrow q$) en la que se establece la veracidad condicionada del consecuente frente al antecedente. (Puede verse claramente que ésta es la forma descripta para una regla).

Adicionalmente, la negación ($\sim p$) y la equivalencia ($p \Leftrightarrow q$) son relaciones también útiles para vincular proposiciones lógicas.

La siguiente tabla corresponde a las relaciones de verdad para las operaciones lógicas descriptas en un contexto de lógica tradicional (dicotómico):

p	**q**	**$p \wedge q$**	**$p \vee q$**	**$p \Rightarrow q$**	**$p \Leftrightarrow q$**	**$\sim p$**
Verdadero	Verdadero	Verdadero	Verdadero	Verdadero	Verdadero	Falso
Verdadero	Falso	Falso	Verdadero	Falso	Falso	Falso
Falso	Verdadero	Falso	Verdadero	Verdadero	Falso	Verdadero
Falso	Falso	Falso	Falso	Verdadero	Verdadero	Verdadero

En la lógica proposicional tradicional la implicación conecta antecedentes y consecuentes sin representar una relación entre la veracidad de las causas y la veracidad de los efectos. Esto deriva, a menudo, en contradicciones con los sistemas físicos por lo que, más adelante, habremos de proveernos de una definición alternativa que sirva a nuestros fines.

Basándonos en las tablas de verdad anteriores, podemos formular diversas tautologías que nos permitan expresar la implicación lógica.

Ejemplos:

1. $\mu_{p \Rightarrow q} = 1 - \mu_{p \cap \sim q}(x,y) = 1 - \text{mín}[\mu_p(x), 1 - \mu_q(y)]$
2. $\mu_{p \Rightarrow q} = \mu_{(\sim p) \cap q}(x,y) = 1 - \text{máx}[1 - \mu_p(x), \mu_q(y)]$

3. $\mu_{p\Rightarrow q}=1 - \mu_p(x) * (1 - \mu_q(y))$
4. $\mu_{p\Rightarrow q}=mín[1, 1 - \mu_p(x) + \mu_q(y)]$

En la lógica tradicional existen, fundamentalmente, dos maneras de efectuar la inferencia que son el **modus ponens** y el **modus tollens.** El modus ponens es vital en la actividad ingenieril dado que pone claramente a la vista las relaciones causa-efecto (situación que constituye la amplia mayoría de nuestros razonamientos y deducciones).

El *modus ponens* se construye en base a dos premisas, una es tenida por verdadera, mientras que la otra es una implicación condicionada a la primera.

Formalmente expresado:

1er premisa: El hecho X es verdadero
2da premisa: **SI** el hecho X es verdadero **ENTONCES** el hecho Y también lo es.
Consecuencia: Y es verdadero

Viendo desde otro punto podemos proponer:

Sea A una determinada característica, que el hecho *x* puede presentar o no, y sea B una consecuencia que se desprenda lógicamente de un *x* con la característica A. Luego el **modus ponens** para esta situación se expresa como sigue:

1er premisa: *x* es A
2da premisa: **SI** *x* es A **ENTONCES** *y* es B
Consecuencia: *y* es B

La expresión simbólica del **modus ponens** es: $\mathbf{(p \wedge (p \Rightarrow q)) \Rightarrow q}$

La lógica difusa adopta nociones de la lógica clásica, pero reemplaza las funciones bivalentes de membresía por funciones difusas de membresía. De este modo la regla "**SI** *u* es A **ENTONCES** *v* es B", donde $u \in U$ y $v \in V$, tiene una función de pertenencia $\mu_{A\Rightarrow B}(x, y) \in [0, 1]$, midiendo el grado de veracidad de la relación de implicación entre *x* e *y*.

En lógica difusa el *modus ponens* se extiende al *modus ponens generalizado* que se expresa como:

1er premisa: *u* es A*
2da premisa: **SI** *u* es A **ENTONCES** *v* es B
Consecuencia: *v* es B*

Mientras que en un sistema de lógica tradicional, el disparo de cada regla está condicionado excluyentemente a la verdad o falsedad absoluta de los antecedentes, en el caso difuso basta un grado de pertenencia no nulo para que esto ocurra, obteniéndose una consecuencia con un grado de similitud no nulo respecto del consecuente de la regla en cuestión. De un modo coherente con la inferencia clásica, $\mu_{B^*}(y)$ se obtiene como:

$$\mu_{B^*}(y)= \mu_{A^*}(x) \circ \mu_{A\Rightarrow B}(x, y)$$

que, adoptando **máximo-estrella** como norma de composición, resulta

$$\mu_{B^*}(y)= máx_{(x\in A)} [\mu_{A^*}(x) * \mu_{A\Rightarrow B}(x, y)]$$

pudiendo, a su vez, adoptarse la relación mínimo() o producto( ) para el operador estrella( ). Si bien esta elección no hace gran diferencia en el comportamiento de SLD aplicados a ingeniería, sí lo hace la forma de la implicación. No cualquier forma de implicar modela adecuadamente el comportamiento causal de un sistema físico. Por esta razón algunos

autores (Mamdani, Larsen) han propuesto implicaciones alternativas como la implicación mínimo o la implicación producto:

$$\mu_{A\Rightarrow B} \equiv \text{mín}(\mu_A(x) , \mu_B(y)) \qquad \text{(Mamdani)}$$

$$\mu_{A\Rightarrow B} \equiv \mu_A(x) \cdot \mu_B(y) \qquad \text{(Larsen)}$$

Estas implicaciones son de baja complejidad computacional comparada con la de las implicaciones más tradicionales. Se debe notar que el comportamiento lógico de estas modalidades de implicación no coincide exactamente con el de la implicación tradicional dado que:

p	q	p => q	p * q	min(p, q)
Verdadero	Verdadero	Verdadero	Verdadero	Verdadero
Verdadero	Falso	Falso	Falso	Falso
Falso	Verdadero	Verdadero	Falso	Falso
Falso	Falso	Verdadero	Falso	Falso

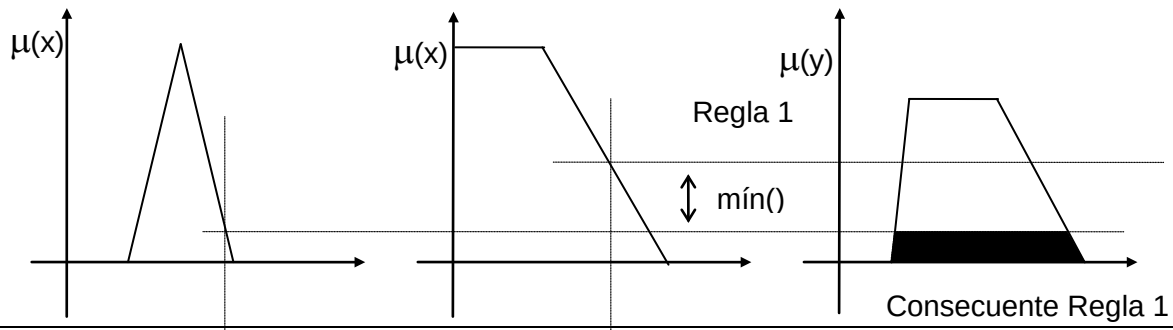
Por esta causa se suele denominar a estas implicaciones restringidas como implicaciones de ingeniería. En esta tabla obviamente hemos asociado números a los valores de verdad para poder evaluar las expresiones (p* q) y min(p, q). Tales valores son Falso=0 y Verdadero=1.

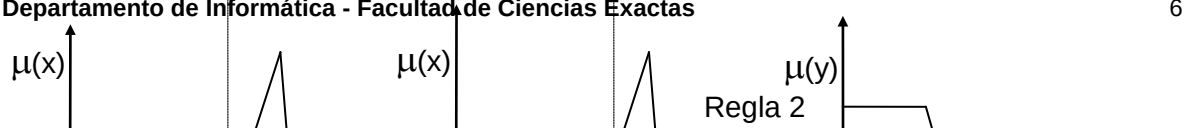
En general un SLD se compone de una cantidad más o menos importante de reglas, contando cada una con uno o más antecedentes y un consecuente. Los valores que asumen los antecedentes están relacionados con las magnitudes numéricas de las respectivas entradas y de las funciones de pertenencia que caracterizan su clasificación lingüística. El proceso de fuzzificación es, efectivamente, el mapeo entre las variables de entrada y sus valores de membresía en esa clasificación.

Para cada regla, la combinación de estos valores de acuerdo al operador “estrella( )” adoptado para componer los antecedentes, arroja un parámetro que es el grado de disparo de la regla. Cada regla disparada da por resultado el subconjunto difuso de su consecuente, (afectado por el valor de activación). Dado que cada regla está vinculada con las restantes mediante una relación inclusiva - o se puede transformar la base de reglas en esta forma -, el conjunto difuso representante de la inferencia no es sino la composición inclusiva de todos los subconjuntos difusos obtenidos como resultado inferencial de cada regla. Con este último, el proceso de defuzzificación se encarga de obtener un único valor numérico representativo del resultado de la inferencia.

Gráficamente, un sistema de dos reglas con dos antecedentes cada una daría lugar a una inferencia de acuerdo al siguiente mecanismo (ver Figura 3) [7].

Las dos reglas tienen a los mismos antecedentes y consecuentes. Los valores, previa *fuzzificación*, resultan en pertenencias no nulas a dos, o más, subconjuntos difusos de entre los que describen lingüísticamente a cada variable. Para cada regla, el mínimo valor entre las membresías de los antecedentes es el que modula el subconjunto consecuente de la misma. Finalmente la combinación mediante el operador máximo( ) de los consecuentes modulados es el subconjunto difuso inferido. Este valor, como explicaremos luego, será defuzzificado para obtener un valor numérico concreto.





5. Defuzzificación

La *defuzzificación*, es la última etapa del proceso. Su función es convertir un valor numérico concreto la salida difusa resultante de nuestro proceso de inferencia.

Existe una gran variedad de métodos para realizar esta actividad, pero no todos resultan realísticos a la hora de su implementación cuando hay compromisos de tiempo, espacio de direccionamiento y costos. Como nuestro sistema basa su inferencia en funciones de membresía que no varían en el tiempo [8] es posible calcular 'a priori' el centro de gravedad de las mismas, en adelante "centroide", como la abcisa que deja la mitad de la integral de dicha función a cada uno de sus lados. Estas razones nos han llevado a elegir una defuzzificación de tipo pesada no ponderada.

Dicha conversión considera la veracidad de cada regla como el 'peso' del centroide de salida de la misma. Su expresión matemática es:

$$y = \frac{\sum_{k=1}^{N} y_k \bullet \mu_k}{\sum_{k=1}^{N} \mu_k}$$

En dicha fórmula :

y_k :es el centroide de la función de membresía de salida asociada a la k-ésima regla.(ver Figura 4)
μ_k :es el grado de realidad de la k-ésima regla para los antecedentes existentes al momento de inferir.
y :es el valor 'crisp' de salida que produce nuestro sistema.

6. Kernel Difuso

El desarrollo de un kernel difuso para microcontroladores estándar con restricciones de ancho de palabra, direccionamiento, etc., empleando herramientas de desarrollo de software con soporte para lógica difusa, se encuentra muy difundido en la actualidad. Sin embargo, la complejidad computacional del empleo de lógica difusa para la resolución de problemas en el campo de la ingeniería, requiere de una gran cantidad de operaciones vectoriales sencillas (sumas, productos, comparaciones, cocientes, etc.), y tiene limitaciones en el tiempo de ejecución.

En este trabajo se presenta un método generalizado para la implementación de un kernel difuso que consiste en una base de datos que relaciona entrada y salida. La operación se realiza vía un acceso directo, que asegura un cálculo difuso de alta velocidad.

Para explicar la concepción del kernel, partiremos definiendo los componentes básicos de nuestro desarrollo.

Sea un vector de entrada n-dimensional **X**:$\{x_1,x_2,...x_n\}$ y sus respectivos conjuntos de funciones de membresía asociados $\mu_j(x_i) = \{ \mu_1(x_i)\ \mu_2(x_i) \ ...\ \mu_m(x_i)\}$.

Cuando un conjunto $\mu_j(x_i)$ es activado por el vector **X**, se genera otro conjunto acotado de posibles valores de salida. Este proceso es tal que dado un valor particular de **X** el conjunto activado será siempre el mismo y la salida para ese valor también lo será [8].

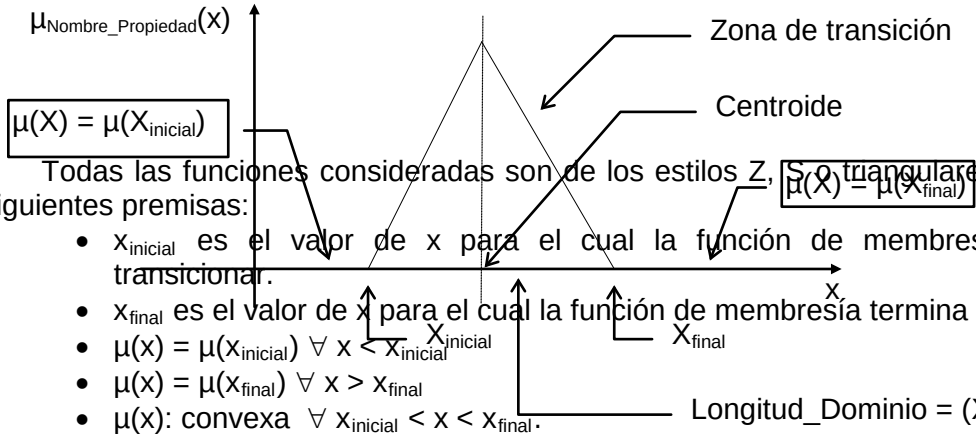


Figura 4

Todas las funciones consideradas son de los estilos Z, S o triangulares y cumplen las siguientes premisas:

- $x_{inicial}$ es el valor de x para el cual la función de membresía comienza a transicionar.
- x_{final} es el valor de x para el cual la función de membresía termina de transicionar.
- $\mu(x) = \mu(x_{inicial})\ \forall\ x < x_{inicial}$
- $\mu(x) = \mu(x_{final})\ \forall\ x > x_{final}$
- $\mu(x)$: convexa $\forall\ x_{inicial} < x < x_{final}$.

Por lo tanto sólo es necesario almacenar la zona transicional de las funciones de membresía (ver Figura 4).

Denominaremos como "zona transicional" de una variable de entrada a la unión de las zonas de transición de cada una de las funciones de membresía de dicha variable.

Siempre y cuando las zonas transicionales sobre el dominio de cada componente de **X** no resulten demasiado extensas, estaremos en la posibilidad de definir una matriz n-dimensional acotada que contenga todos los valores de salida. Las dimensiones de dicha matriz están asociadas a cada uno de los vectores que generan el subespacio vectorial, contenido en N^n (N^n: espacio n-dimensional de números naturales) en el cual se encuentra **X**. Para acceder a los elementos de la matriz podemos operar de forma directa con los valores de **X** muestreados (a menos de una traslación en N^n). Para ello, realizamos una transformación de la matriz en un arreglo unidimensional, que se accede por medio de un puntero calculado de acuerdo a lo siguiente:

Sea $x'_k = G(x_k)$, donde G() es un operador que mapea la zona transicional de la variable x_k en (m_k+1) valores enteros según:

$$G(x): \begin{cases} 0 & x < x_{inicia} \\ \Xi\left(\left(x - x_{inicial} \Big/ x_{final} - x_{inicial}\right) \bullet m_k\right) & x_{final} \geq x \geq x_{inicial} \\ m_k & x > x_{final} \end{cases}$$

donde $\Xi(\ )$ es la parte entera redondeada exactamente.

y sean:

$$x'_k : \{0,1,.....,m_k\}$$

$$L\arg o\left(x'_k\right) = m_k + 1$$

en consecuencia:

$$Ptr = \frac{\sum_{k=1}^{n}\left(x'_k \bullet \prod_{i=1}^{k} L\arg o\left(x'_i\right)\right)}{L\arg o\left(x'_1\right)}$$

dará un mapeo como el requerido.

El costo de este método en términos de memoria insumida es:

$$\text{Memoria ocupada (bytes)} = \prod_{k=1}^{n} L\arg o\left(x'_k\right)$$

7. Aplicación Práctica

El objetivo central de este trabajo es el desarrollo de un *kernel difuso* dedicado que permita extender el uso de microcontroladores estándar con restricciones de ancho de palabra, direccionamiento, etc., a aplicaciones más complejas. Para ello se trabajó sobre un modelo simple que es el control de la velocidad de un motor de corriente continua (estabilización ante cargas mecánicas), disponiendo de un mínimo de conocimiento de sus características.

El actuador elegido es un transistor empleado como llave de c.c. que se controla mediante una onda de 10mSeg de período aplicada en la base, cuyo duty cycle es función de la carga y velocidad requeridos.

La medición de velocidad se efectuó mediante un tacómetro digital conformado por un conversor frecuencia/tensión y un conversor A/D. La entrada de este módulo es un tren de pulsos proveniente de un disco ranurado solidario al eje del motor. El conversor frecuencia/tensión se realiza mediante un PLL cuya señal de control es apropiadamente acondicionada para entrar al conversor A/D. La resolución obtenida es del 0.39% de fondo de escala (aprox. 2300 R.P.M.).

El microcontrolador empleado para esta implementación es un 8031 @ 11 Mhz. Éste debe llevar a cabo las siguientes tareas (Ver figura 5):

- medición de la velocidad del motor
- generación de las variables de entrada al kernel difuso
- inferencia difusa
- actualización del pwm según la inferencia lo indique

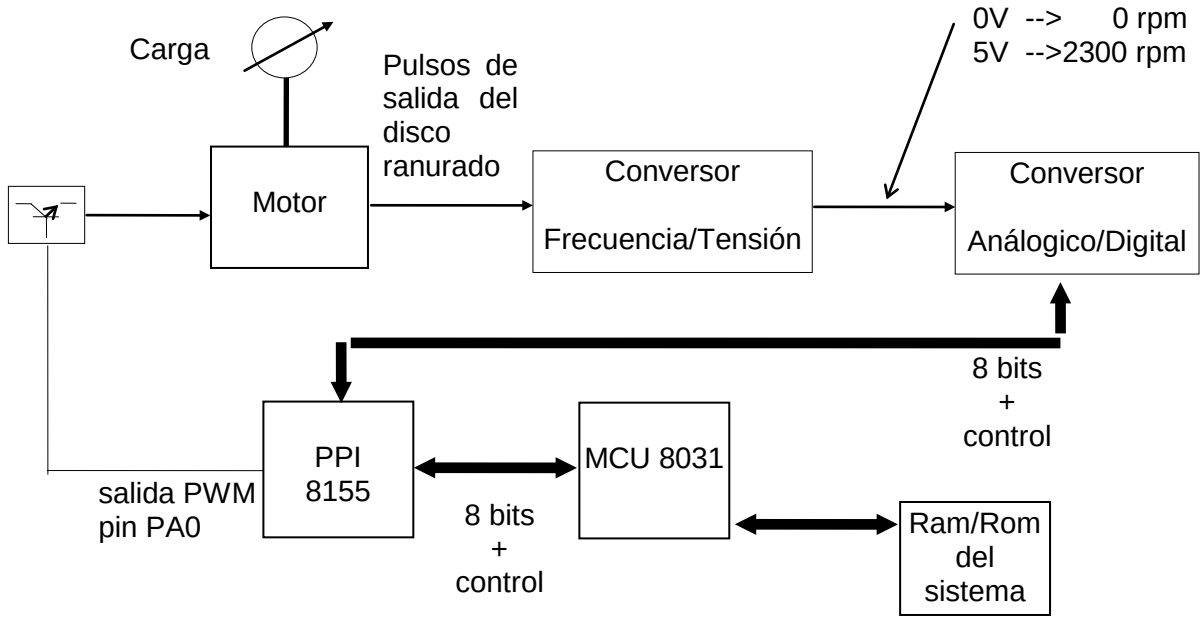


Fig. 5 - Hardware

El valor de velocidad adquirido es transformado en sendas entradas al kernel difuso del siguiente modo:

1. Error de velocidad = (Velocidad medida - Velocidad deseada) + 127
2. Incremento de velocidad = (Velocidad medida(t_n)-Velocidad medida (t_{n-1})) +127.

Ambas magnitudes están en un rango de 0 a 255.
La salida del kernel difuso es un incremento sobre el duty cycle del PWM que puede variar entre -10 y +10.

La base lingüística empleada [6] es la que se describe en la siguiente tabla:

	Si		
Regla nº	Error_Veloc es	Incr_Veloc es	Entonces
1	Cero	Cae	Subir
2	Cero	Quieto	Dejarlo
3	Cero	Sube	Bajar
4	Exceso_Chico	Cae	Dejarlo
5	Exceso_Chico	Quieto	Bajar
6	Exceso_Chico	Sube	Bajar
7	Defecto_Chico	Cae	Subir
8	Defecto_Chico	Quieto	Subir
9	Defecto_Chico	Sube	Dejarlo

10	Defecto_Grande	Cae	Subir_Bastante
11	Defecto_Grande	Quieto	Subir_Bastante
12	Defecto_Grande	Sube	Subir
13	Exceso_Grande	Cae	Bajar
14	Exceso_Grande	Quieto	Bajar _Bastante
15	Exceso_Grande	Sube	Bajar _Bastante

Las funciones de membresía asociadas a los conjuntos de la tabla anterior se muestran en la Figura 6. Los números referencian las siguientes funciones de membresía:

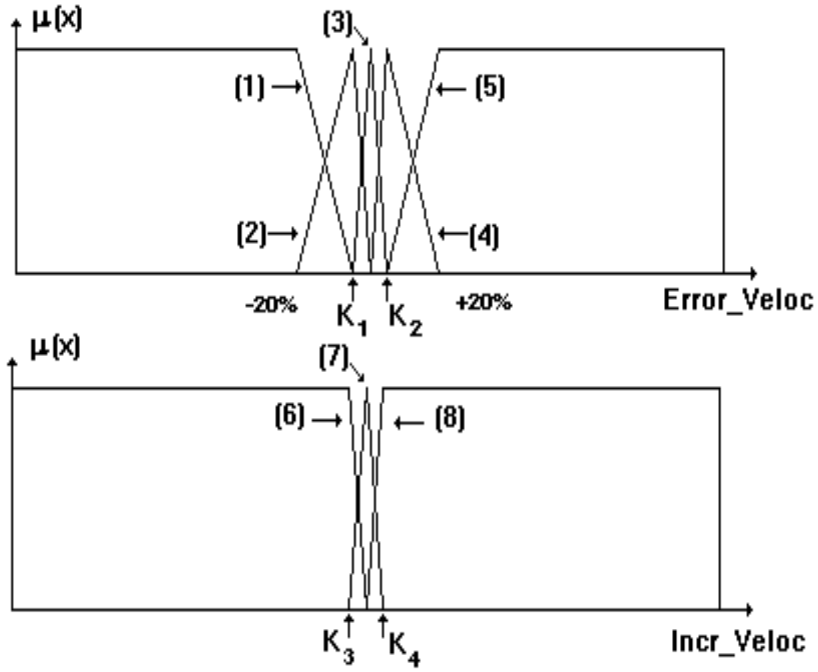


Figura 6 - Funciones de membresía

1. Defecto_Grande
2. Defecto_Chico
3. Cero
4. Exceso_Chico
5. Exceso_Grande
6. Cae
7. Quieto
8. Sube

Como de las funciones de membresía de salida lo único que empleamos son los centroides, daremos sus valores en la siguiente tabla:

Nombre de la función de salida	Valor del centroide
Subir_Mucho	+5
Subir	+3
Dejarlo	0
Bajar	-2
Bajar_Mucho	-6

El software está compuesto de dos grandes bloques. Uno opera con variables reales de entrada/salida y el otro implementa el control difuso de la velocidad (ver Figura 7).

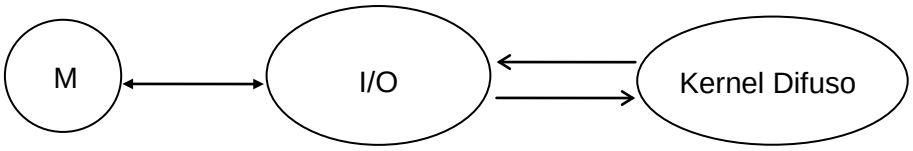


Figura 7

8. Ensayos y Mediciones

En esta sección mostramos los resultados obtenidos através de la experiencia práctica de controlar el par motor-generador aplicando el método descripto en la sección 6.

En principio se realizaron medidas de la variación del espacio de memoria requerido al variar la zona transicional de las entradas, como se aprecia en la siguiente tabla:

Memoria	Zona de Transición Error_Veloc	Zona de Transición Incr_Veloc
(51*13)= 663 bytes	porcentual: [-20%; +20%] valores de código: [101; 152]	porcentual: [-5%; +5%] valores de código: [121; 133]
(51*27)= 1377 bytes	porcentual: [-20%; +20%] valores de código: [101; 152]	porcentual: [-10%; +10%] valores de código: [114; 140]

Como se ve la cantidad de memoria requerida es muy razonable.

Luego se realizaron mediciones de la velocidad del motor por medio de una herramienta de software desarrollada a tal efecto, que nos permitió visualizar gráficamente el comportamiento del sistema. Las figuras 8, 9, 10 y 11 muestran los resultados obtenidos para: a) dos escalones de carga y b) evolución de rotor bloqueado a vacío, para distintos porcentajes de las zonas de transición de error de velocidad e incremento de velocidad. En ellas la escala de velocidad 'v(t)' es (1/18.11) [r.p.m.] y el setpoint del sistema está situado en 75*18.11=1358.25[r.p.m.]. Por su parte las escalas de tiempo son de lectura directa.

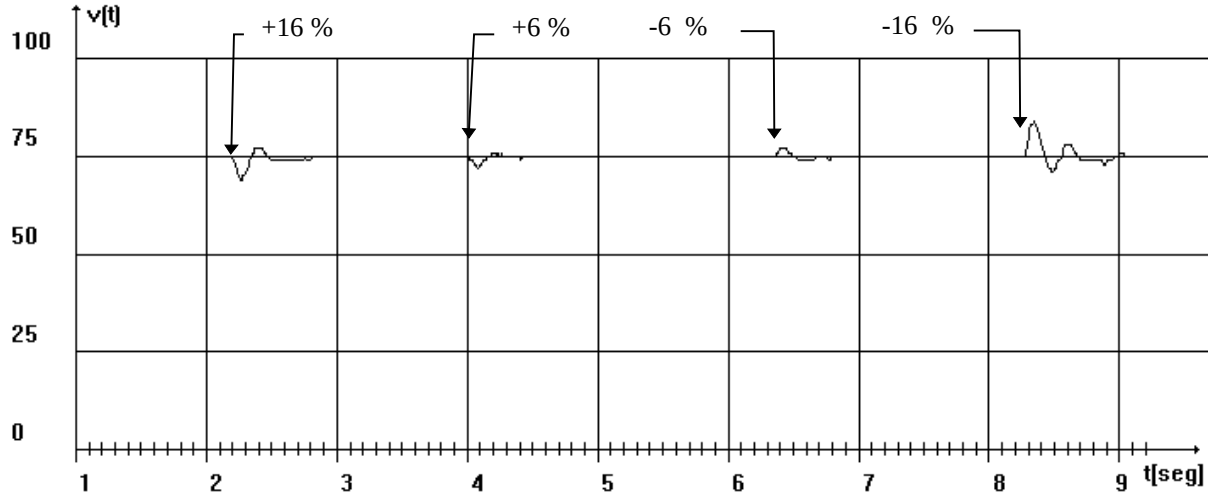


Figura 8.a : K1,K2=±5% ,K3,K4=±5%(ref fig 6)

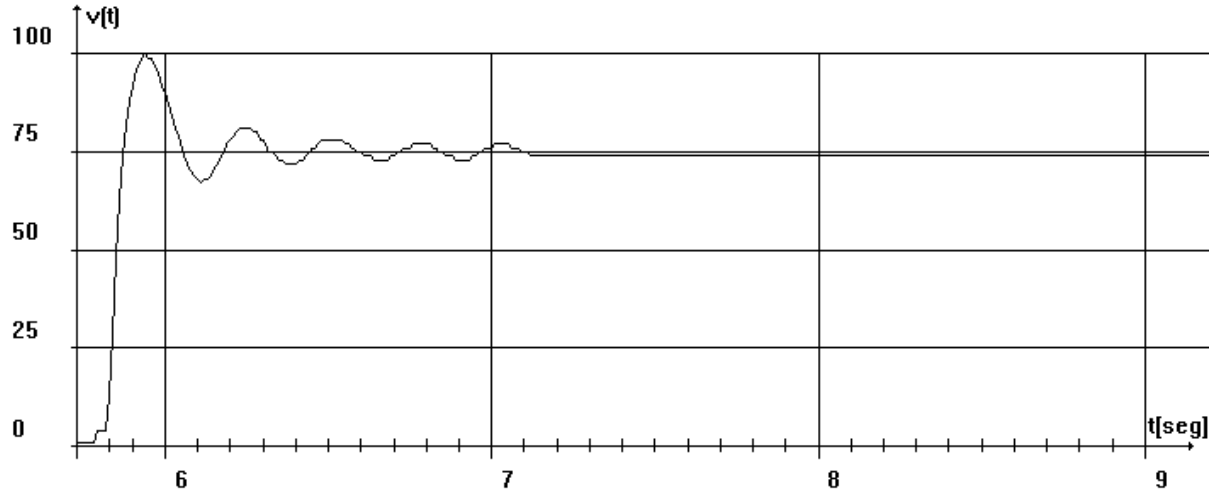


Figura 8.b: K1,K2=±5%,K3,K4=±5% (ref fig 6)

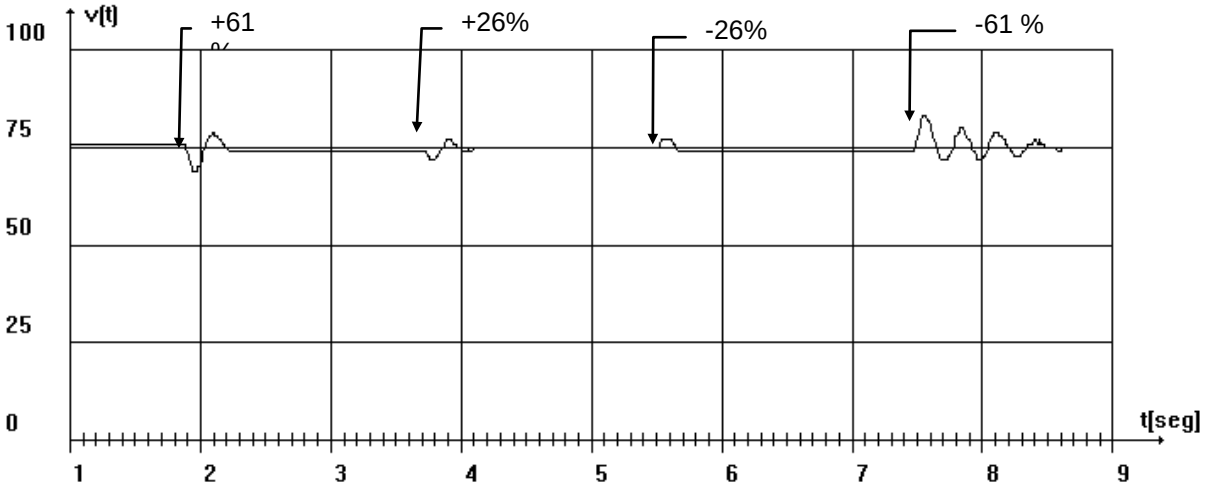


Figura 9.a: K1,K2=±5%,K3,K4=±10% (ref fig 6)

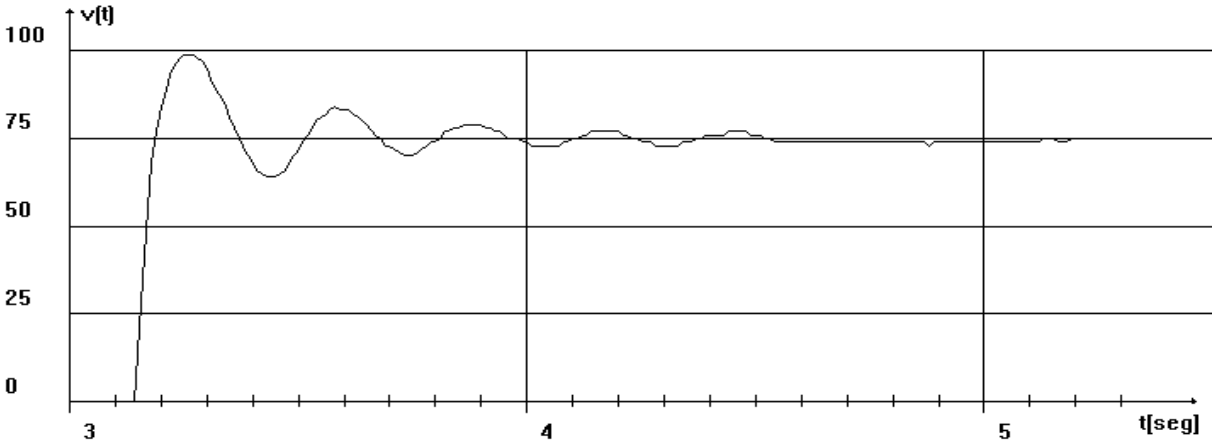


Figura 9.b: K1,K2=±5%,K3,K4=±10% (ref fig 6)

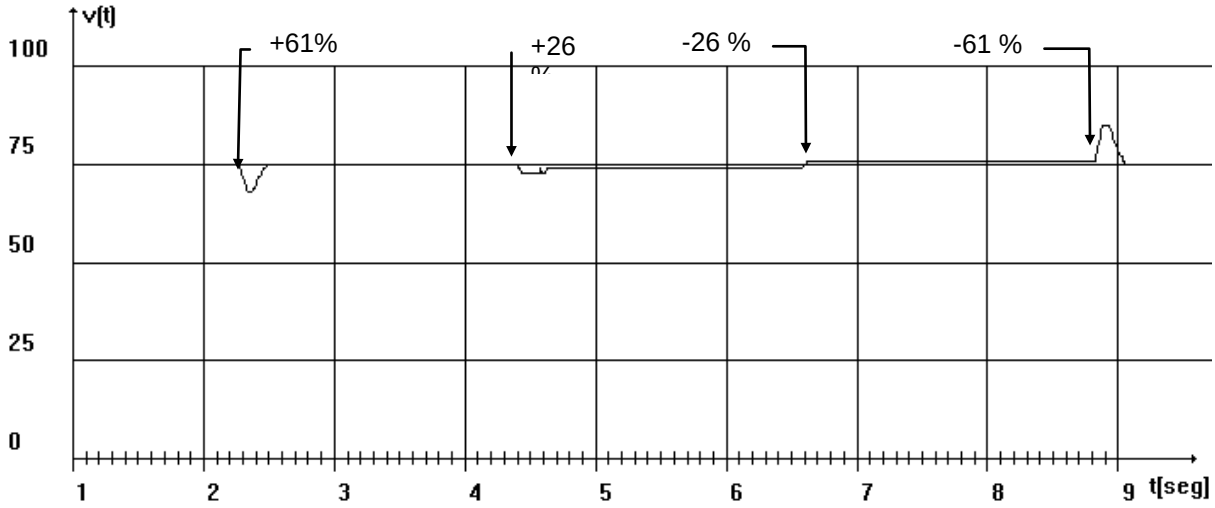


Figura 10.a: K1,K2=±10%,K3,K4=±5% (ref fig 6)

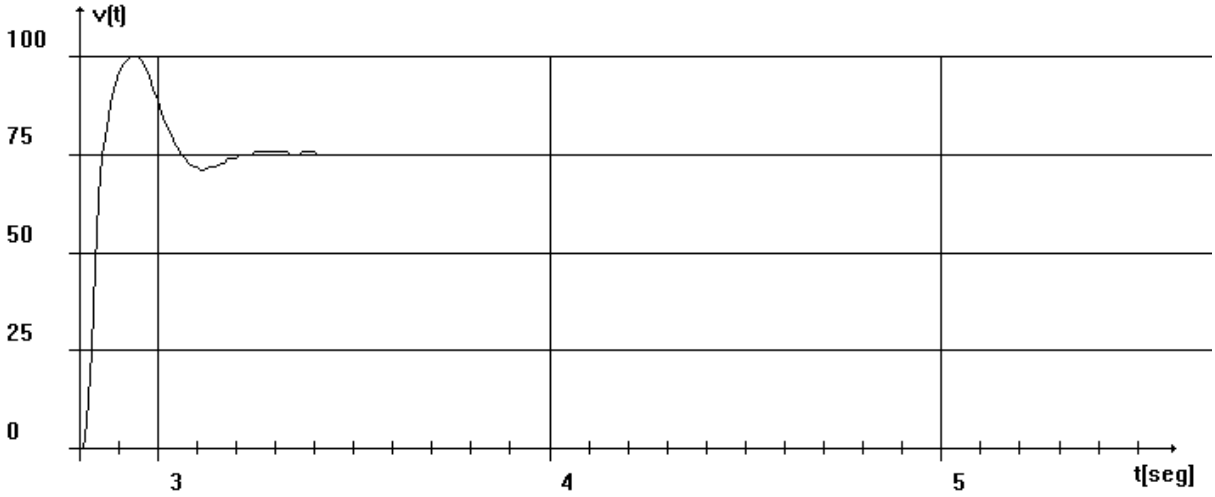


Figura 10.b: K1,K2=±10%,K3,K4=±5% (ref fig 6)

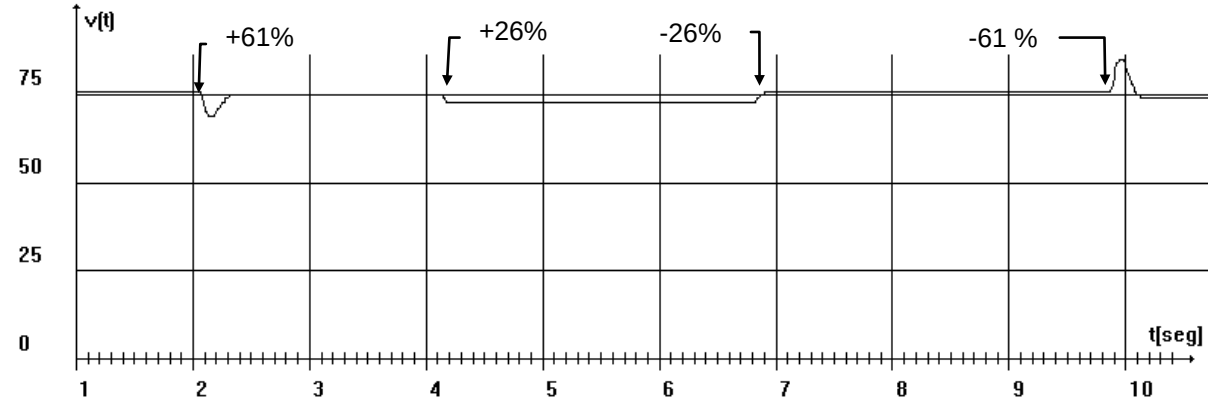


Figura 11.a: K1,K2=±10%,K3,K4=±10% (ref fig 6)

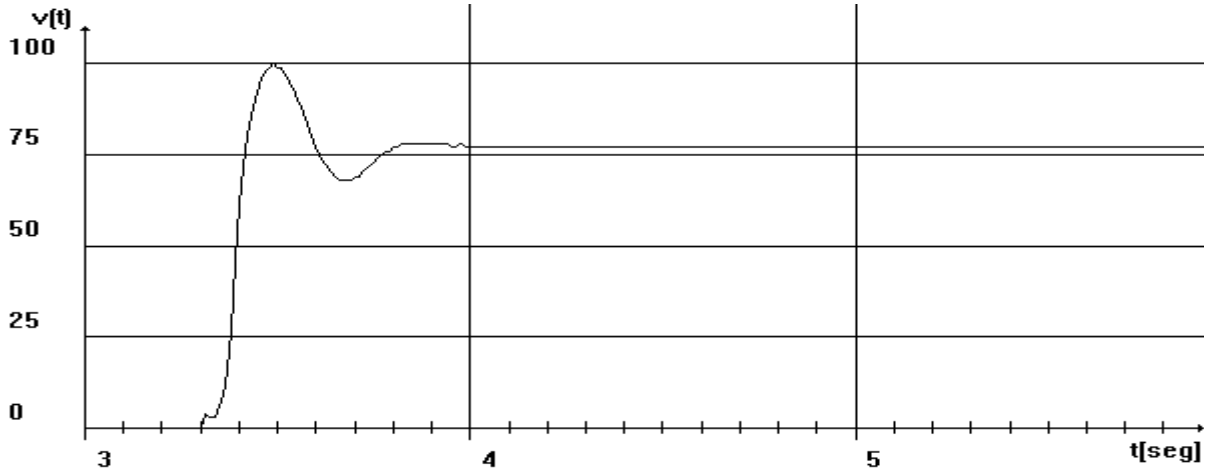


Figura 11.b: K1,K2=±10%,K3,K4=±10% (ref fig 6)

Observando las gráficas de las figuras 8 a la 11 se puede observar que:

- cuando la tolerancia del error de velocidad se incrementa, se disminuyen los transitorios y se incrementa el porcentaje de error en el estado estacionario.
- las figuras 8.b y 10.b no son contradictorias en relación con el punto anterior, sino que el sistema se considera estable dentro de una banda de tolerancia.
- todos los tiempos de estabilización se encuentran por debajo de los 400 mseg (peor caso).

Por último, se graficó un diagrama de tiempos de ejecución de las diversas tareas, donde se aprecia que el tiempo del kernel es realmente pequeño a todo fin práctico (ver Figura 12).

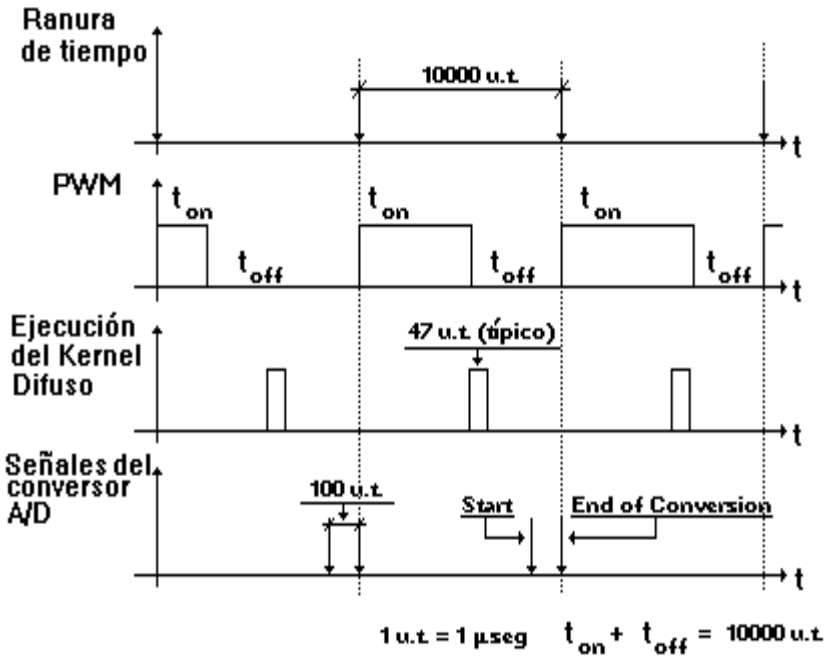


Figura 12

9. Conclusiones

En primera instancia se podría mencionar la cualidad que un SLD tiene para resolver una aplicación sin tener el conocimiento de un modelo matemático preciso.

En este trabajo se han desarrollado los conceptos básicos para implementar un kernel difuso, consistente en una base de conocimeintos que relaciona en forma directa las variables de entrada “crisp” con la salida, permitiendo un cómputo difuso de alta velocidad, aún corriendo sobre una plataforma con numerosasa restricciones. Esto ha dado la posibilidad encarar soluciones complejas con elemnetos de cómputo de bajo costo, sin tener necesidad de recurrir al empleo de hardware orientado para el procesamiento difuso.

El reducido tiempo de ejecución de nuestro kernel difuso y un estudio del resto de las actividades a cubrir, nos muestran que es factible disminuir el tiempo de ranura hasta unos 300 μseg. Este método es una interesante opción a la hora de encarar soluciones de hardware para sistemas particularmente exigentes en cuanto a velocidad de operación y reacción (por ejemplo el control inteligente de una fuente conmutada dc-dc).

Otra aplicación realmente interesante es el control simultáneo de varios dispositivos ‘lentos’ por medio de un único microcontrolador (si la disposición física de los dispositivos así lo permite). La ventaja en este caso será una ostensible reducción de costos de hardware y de mantenimiento.

El costo de éste método se verá afectado al incrementar el número de variables de entrada y/o las zonas transicionales de una o más variables, impactando directamente sobre la cantidad de memoria requerida.

La cantidad de memoria necesaria es independiente del número de funciones de membresía y reglas definidos, una vez que las zonas transicionales de cada variable hayan sido delimitadas. Por lo general, de acuerdo a la definición del experto, dichas zonas se pueden mantener acotadas. Ésta es una situación muy usual en la problemática ingenieril.

10. Referencias

[1] Mendel, "Fuzzy Logic System for Engineering: A Tutorial", Proceedings of the IEEE, Vol. 83, Nº 3, March 1995.
[2] Costa A., De Gloria A., Faraboschi P., Pagni A and Rizzotto G., "Hardware Solutions for Fuzzy Control", Proceedings of the IEEE, Vol. 83, Nº 3, March 1995.
[3] Kosko Bart, "Neural Networks and Fuzzy Systems"Prentice-Hall, INC:1992.
[4] Microchip, "fuzzyTECH-MP Explorer, User's Guide", 1994.
[5] Brignole Diana, "Lógica y Teoría de los Conjuntos Difusos", Dto. Matemática UNS, 1996.
[6] Guillemin P., "Universal Motor Control with Fuzzy Logic", Fuzzy Set and Systems, Volume 63, Number 3, May 10, 1994.
[7] JYH-Shing R. and Chuen-Tsai Sun, "Neuro-Fuzzy Modeling and Control", Proceedings of the IEEE, Vol. 83, Nº 3, March 1995.
[8] Kartalopoulos S. "Understanding Neural Networks and Fuzzy Logic", IEEE Press, 1996.